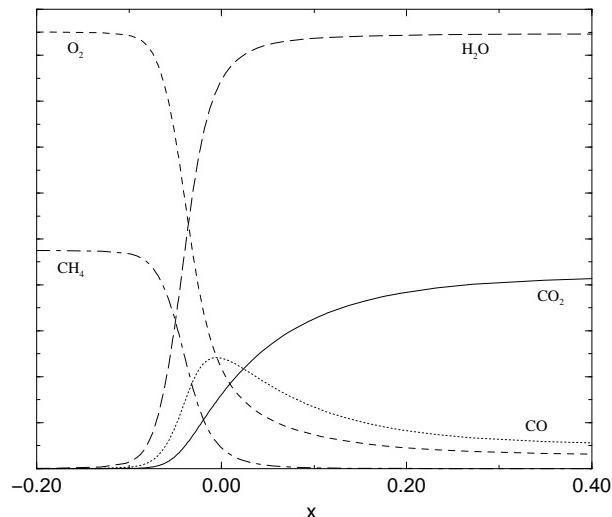


EPORCK V1.8 User's Guide

Charles MARTIN

May 19, 2004



Evolutionary Procedure for the Optimisation of Chemical Kinetics

Introduction

EPORCK has been developed in order to ease the fitting procedure of reduced¹ chemical schemes. This is achieved by automating the optimisation procedure. Thus EPORCK is devoted to finding scheme parameters' set that fits the user's requirements. EPORCK is a **GA** (Genetic Algorithm) based software. Nevertheless future developments will include Gradient based methods in order to accelerate final convergence.

¹less than five reactions

Contents

1	Methodology of GA applied to chemical scheme fitting	1
1.1	Chemical scheme definition	1
1.2	The optimisation problem: cost function minimisation	1
1.3	GA basic principles	2
2	EPORCK architecture and requirements	4
2.1	Mainframe	4
2.2	Reference criteria and cost function	5
2.3	EPORCK features	5
3	EPORCK input files description	7
3.1	scheme.c file	8
3.2	input_genocop.dat file	10
3.3	input_eporck.dat file	11
3.4	restart.eporck file	12
3.5	equil.job file	12
3.6	Script shell files	12
4	Running EPORCK	13
4.1	Run check list	13
4.2	Getting started	13
4.3	Output files	14
4.4	Specific tools	18
5	Problem sample	19

1 Methodology of GA applied to chemical scheme fitting

This section describes the different methods used in EPORCK .

1.1 Chemical scheme definition

Parameters : $A Y_F^{nF} Y_O^{nO} \exp(-E_a/RT)$	A [CGS]	n_F [-]	n_O [-]	E_a [cal/mol]
Reaction #1	A_1	n_{F1}	n_{O1}	E_{a1}
...
Reaction #N	A_N	n_{FN}	n_{ON}	E_{aN}

Figure 1: Chemical scheme representation

Chemical schemes are characterised for each reaction (see Fig. 1) by a four real numbers set. This allows to define the Arrhenius law. Thus, a N-step chemical scheme requires to fit $4N$ parameters.

1.2 The optimisation problem: cost function minimisation

The fit of scheme parameters can be seen as an optimisation problem. The aim is to find the complete parameters set that let the corresponding scheme fits some predefined reference outputs. This kind of problem are often reduced to a minimisation one. The idea is to minimise a cost function that represent the error or the distance between the reference (or target) scheme's outputs and actual ones. The function to minimise is presented in Sec. 2.2

The set of methods that solve the optimisation problems can be split into two main types:

- Gradient based methods
- **GA** based methods

Gradient based methods are theoretically more efficient, and less CPU-time consuming. But global minimum research needs to use special “multiple seeding” technics, since their main feature is to find local minima. The main drawback of these methods is their poor robustness face to the difficulty to sometimes evaluate the function to minimise. In other word when the evaluation of the cost function fails, the classical gradient methods are unable to evaluate function gradients and then to step beyond the actual phase space position. Additionally these methods are not suitable when the cost function shape is noisy or nearly chaotic.

On the other hand , **GA** methods handle very well unconverged function evaluations, and are very robust in a general sense. Their main advantage resides in the way they

balance **domain exploration** (i.e research of new solutions) and **optimum determination** (precise location of the minimum). If this balance is well established, they are quite unsensitive to initial conditions. Their main drawback is perhaps the number of function evaluations face to efficient gradient methods. This evaluation cost and general convergence may be greatly affected by **GA**'s parameters tuning.

1.3 GA basic principles

We introduce some basic considerations about **GA**. A **GA** is used as a minimiser. Fig. 2 illustrates the direct relation between real life parameter (the genes) and our problem. For our fit, we deal with a “monochromosomal animal” whose genes are the scheme parameters. The biological vocab is naturally used to describe **GA**. For our problem we can make the following bijection:

- a chromosom \iff a chemical scheme representation
- a gene \iff a chemical scheme parameter
- an individual \iff one chemical scheme
- a population \iff a set of chemical schemes
- a generation \iff a set of “breded” schemes

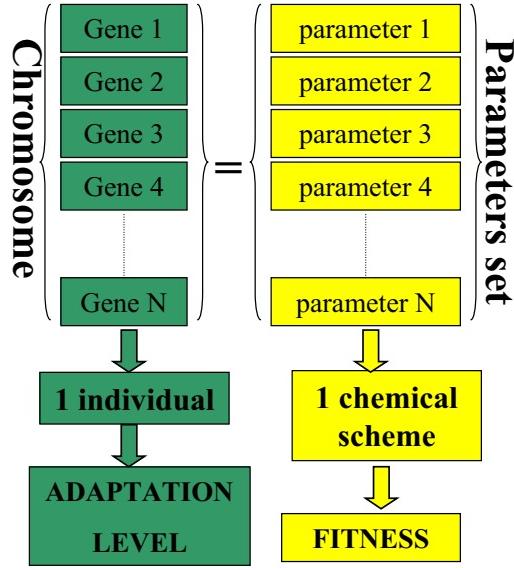


Figure 2: Animal genes vs scheme parameters set

GA and more generally the **evolutionary processes** are centered onto an evolutionary loop (Fig. 3). This iterative procedure lets evolve a population generation by generation under a **selective** pressure, i.e. keep alive individuals that fits our requirements. The selective criterion is usually called **fitness function** or **objective function**.

In our case, the fitness is kind of distance between the individual and a target point in the search space. So be carreful not to do some confusion, small value of the fitness function caraterises a well fitted individual. For this reason the fitness function term is often replaced here, by **cost function**. New individulals are the offspring of a single parent mutation or a cross-over between a couple of parents. The balance between domain exploration (Fig. 3 GENETIC OPERAORS block) and optimum determination (Fig. 3 SELECTION block) is one of the key to success.

- Strong selective pressure, by decrasing population diversity leads to premature convergence i.e the minimum found is only local.
- Weak selective pressure and too much mutation in population slow down optimum determination by wasting (CPU-) time in the evaluation of unfitted individuals (distant form the optimum). The search is uneffective.

The other main key to succes in **GA** performing, is the definition of the cost function. This definition determines directly the shape of the search space hyper-surface, so may greatly influence the convergence rate of **GA** methods. Actually the cost function represents a norm of the distance that seprates the individual from a reference point. This reference point is the target to reach. It must be realistic, i.e. reachable. For example, it makes no sense to fit a 1-step scheme on S_L for **both** lean and rich operating points. See 2.2 for our cost function implemetntation.

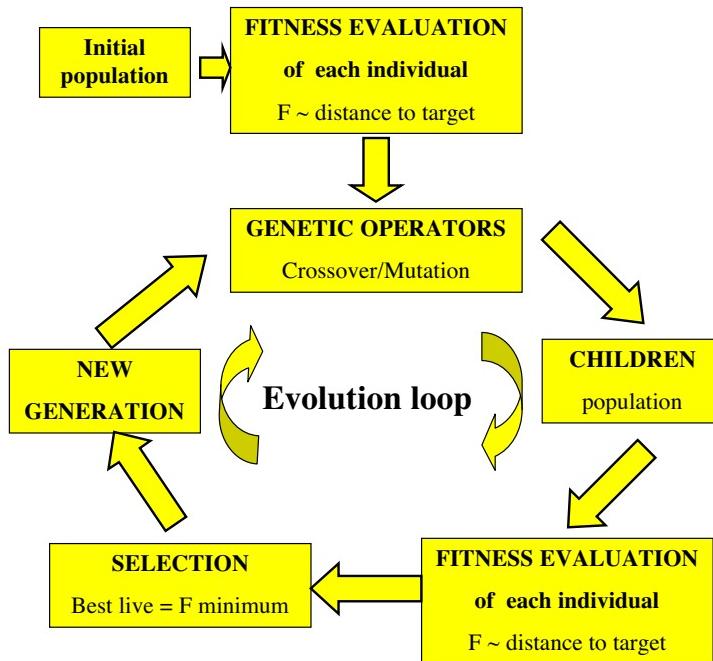


Figure 3: **GA** general principle

2 EPORCK architecture and requirements

2.1 Mainframe

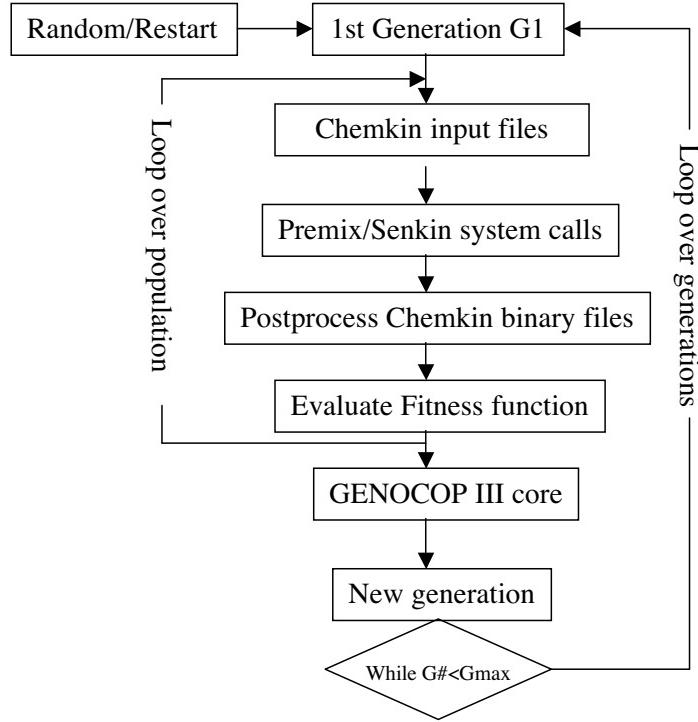


Figure 4: EPORCK general frame

Fig. 4 presents EPORCK structure. The evolutionary loop is initialised by a scheme population. This initial set may be generated randomly respect to parameters bounds, or generated by previous EPORCK run (restart) or pre-existing scheme. Restart as random initialisation may be so called **single** or **multiple**:

- **single**: only one scheme is generated then replicated to obtain an initial population of clones.
- **multiple**: each generated scheme differs *a priori* from its brothers.

Note that EPORCK use a constant population (number of schemes) in the running generation. Once the initial population enters the main loop, each scheme is evaluated (population loop) by PREMIX (or SENKIN in ignition version). Note that PREMIX /EPORCK coupling uses exchange files on disk, so avoid network file system (NFS) that are much slower than local hard disks. Nevertheless this weak coupling does not affect global performance since 95% of CPU-time is consumed by PREMIX . Then the GA core take the population as input to operate on it, selected genetic operators, and selective pressure (some die some live but population remain constant). The main loop is thus closed. Note that the process stops when reaching a specified generation number.

2.2 Reference criteria and cost function

The selectable criteria R_k , are extracted from a laminar flame structure.

- S_l , the laminar flame speed.
- T_{out} , the burnt gas temperature.
- $\delta_L^0 = \frac{T_{out} - T_{in}}{\max(\frac{\partial T}{\partial x})}$, the flame thermal thickness.
- Y_k^{out} , the k^{th} species outlet mass fraction.

The fitness² or cost function F is defined as follows:

$$F = \sum_k \alpha_k F_k$$

where $\{\alpha_k\}$ are the respective weights on the criterion set,

$$F_k = \sum_\phi \alpha_\phi \left| \ln \left| \frac{R_k}{R_k^{ref}} \right| \right|$$

where $\{\alpha_\phi\}$ are respective weights on operating points. $\{R_k^{ref}\}$ are respective reference or target values of selected criteria. **Criteria selection, and weights settings are user defined.**

2.3 EPORCK features

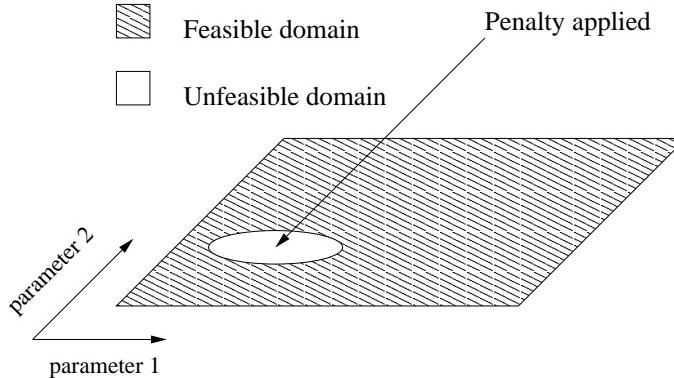


Figure 5: non-conex 2D search space

EPORCK features concerning the definition of the problem:

- multi-objective optimisation capability.
- arbitrary numbers of parameters to fit. (multi-step schemes)

²fitness is the term used by **GA** community, in our case small fitness function = high scheme fitness

- all parameters are upper and lower bounded, must be set by the user. Theoretically, a convex domain³ is needed. Nevertheless unfeasible part(s) of the search space are well handled by EPORCK applying a penalty to unconverged solutions cost function (Fig. 5).
- limited number of target quantities (S_l , T_{out} , δ_L^0 , Y_k^{out}) based on laminar flame structure. (Ignition delays also but in an another EPORCK version)
- arbitrary number of operating points depending on Pressure, Temperature and/or Mixture Composition.

EPORCK features concerning the resolution of the problem:

- arbitrary maximum number of generations
- arbitrary number of individuals per generation
- a selection of seven genetic operators. Simple refers to one gene, whole to the chromosom:

– **simple uniform mutation:**

The operator selects randomly a gene and mute it randomly with a uniform probability distribution.

This operators plays an important role in the early phases of evolution process as the individuals are allowed to move freely within the search space. This operator is essential when starting with a clone population as it introduces novetly (phase space exploration).

– **boundary mutation:**

The operator selects randomly a gene g and mute it to one of its upper or lower boundary. This operator is usefull in early stage of the optimisation when user defined parameters boundary may limit the optimisation process. If the best individual has one of its gene at one boundary, the optimisation is restricted by search space definition.

– **simple non-uniform mutation:**

The operator selects randomly a gene and mute it to g' randomly with a non-uniform probability distribution defined by :

$$g' = \begin{cases} g + \Delta(t, \text{upper_bound} - g) & \text{or (with equal probability)} \\ g - \Delta(t, \text{lower_bound} - g) \end{cases}$$

The function $\Delta(t, y)$ returns a value in the range $[0, y]$ such that the probability of $\Delta(t, y)$ being close to 0 increases as the generation number t increases. This property causes this operator to search the space uniformly initially (exploration) and very locally at later state (focus on the most promising region).

$$\Delta(t, y) = y.r. \left(1 - \frac{t}{T}\right)^b$$

³also convex

where r is a random number from $[0..1]$, T is the maximum generation number, and b is a parameter determining the degree of non-uniformity.. The non-uniform operators are responsible of fine tuning capabilities of the system.

- **whole non-uniform mutation:** This operator has the same behavior than the simple non-uniform mutation. The mutation is applied on the whole chromosom, i.e. each gene.
- **simple arithmetical crossover:** The operator selects randomly a gene location of two individuals g_1 and g_2 . There are two offsprings with the same genome except for the crossing gene location:

$$\begin{aligned} g'_1 &= ag_1 + (1 - a)g_2 \\ g'_2 &= ag_2 + (1 - a)g_1 \end{aligned}$$

This operator uses a random value $a \in [0, 1]$.

Arithmetical crossovers must be selected anyway. They have a stabilisation effect on the evolutionary processes.

- **whole arithmetical crossover:** This operator has the same behavior than the simple arithmetical crossover. The crossover is applied on the whole chromosom, i.e. each gene.
- **heuristic crossover:** This operator is a unique crossover for the following reasons:
 - * it uses values of the objective function in determining a direction of search.
 - * it produces only one offspring, and may produce no offspring at all.

This operator behaves according to the rule:

- * $F(I_2^{parent}) \leq F(I_1^{parent})$ i.e individual 2 is better than 1.
- * $I_{child} = r(I_1^{parent} - I_1^{parent}) + I_2^{parent}$ with r a random number from $[0..1]$.
If after some attempts (varying r), the generated offspring is still not feasible (i.e out of boundaries), then no offspring is produced.

Its major responsibilities are fine local tuning and search in most promising direction.

- adjustable selective pressure parameter. It is not recommended to modify it.
- adjustable parameter for non-uniform mutation operators. It is not recommended to modify it.
- A restart functionality allows to restart an optimisation from the best individual of a previous optimisation. It is also possible to restart from an entire generation. See Sec. 3.4.

3 EPORCK input files description

Here are the input files definition:

- **scheme.c** No parser has been developed yet. Thus PREMIX format scheme definition must be coded by user in C, then linked with EPORCK binaries and CHEMKIN libraries⁴. See Sec. 3.1 for details.
- **input_eporck.dat**: contains information about operating points and also about constraints' number, type and value. Cost function weights are also defined there. See Sec. 3.3 for details.
- **input_genocop.dat**: contains information about parameters' number, range and also population size maximum generation number and genetic operators selection. See Sec. 3.2 for details
- **premix.job**: It is the PREMIX command files. If this file does not exist, it is automatically generated. If premix.job is present, it must be consistent with equil.job and input_eporck.dat settings. i.e. same number of operating points, and operating condition with respect to reference (constraint) value. See Sec. 3.5 for details.
- **equil.job**: It is the EQUIL command files. It must be consistent with operating conditions. It is used to generate runtime the premix.job file. See Sec. 3.5 for details.
- **premix.sh** is a sh script shell file that launch the transport interpreter and PREMIX . See Sec. 3.6 for details.
- **equil.sh** is a sh script shell file that launch the CHEMKIN interpreter and EQUIL . See Sec. 3.6 for details.
- **restart.eporck** optional file that contains chromosome(s) of individual(s) for restart procedure. See Sec. 3.4 for details.

3.1 scheme.c file

X[] is the real vector of parameter to fit (indexed from 1).

```
#include "genocop.h"
#include "extern.h"
void write_scheme(X,file_sel)
/* Dummy : */
VECTOR X;
int file_sel;
{
/* Local : */
char *filename;
char *line;
float rord,nu;
FILE *scheme_file;
if (file_sel==0)
{
  filename="premix.scheme";
}
else
{
```

⁴chemkin.a, chemkin_public.a, surface_chemkin.a

```

        filename="best.scheme";
    }
scheme_file = fopen(filename,"w");
if(scheme_file == NULL)
{
    printf("Open of %s for output failed",filename);
    fclose(input);
    exit(1);
}
rewind(scheme_file);
/** SCHEME FILE EDITION ****/
line="ELEMENTS \n"; fprintf(scheme_file,"%s",line);
line=" H   O   C   N \n"; fprintf(scheme_file,"%s",line);
line="END \n"; fprintf(scheme_file,"%s",line);
line="SPECIES \n"; fprintf(scheme_file,"%s",line);
line="CH4 02 CO C02 H2O N2 \n"; fprintf(scheme_file,"%s",line);
line="END \n"; fprintf(scheme_file,"%s",line);
line="REACTIONS \n"; fprintf(scheme_file,"%s",line);
/* Reaction #1 : */
line="CH4+1.5O2 => CO+2H2O \t\t"; fprintf(scheme_file,"%s",line);
/* write A1 b1 Ea1 */
fprintf(scheme_file,"%E \t %3.1f \t %7.1f \n",pow(10,X[1]),0.0,X[2]);
/* write non stoichiometric coefficients
   CH4 : */
line="FORD / CH4 "; fprintf(scheme_file,"%s",line);
fprintf(scheme_file,"%f /\n",X[3]);
/* O2 : */
line="FORD / O2 ";fprintf(scheme_file,"%s",line);
fprintf(scheme_file,"%f /\n",X[4]);
/* Reaction #2 : */
line="CO+0.5O2 = CO2 \t\t";fprintf(scheme_file,"%s",line);
/* write A2 b2 Ea2 */
fprintf(scheme_file,"%E \t %3.1f \t %7.1f \n",pow(10,X[5]),0.0,X[6]);
/* write non stoichiometric coefficients
   CO :*/
line="FORD / CO "; fprintf(scheme_file,"%s",line);
fprintf(scheme_file,"%f /\n",X[7]);
/* warning, impose reverse order to maintain equilibrium */
nu=-1.0;
rord=X[7]+nu;
line="RORD / CO ";fprintf(scheme_file,"%s",line);
fprintf(scheme_file,"%f /\n",rord);
/* O2 : */
line="FORD / O2 "; fprintf(scheme_file,"%s",line);
fprintf(scheme_file,"%f /\n",X[8]);
/* warning, impose reverse order to maintain equilibrium */
nu=-0.5;
rord=X[8]+nu;
line="RORD / O2 ";fprintf(scheme_file,"%s",line);
fprintf(scheme_file,"%f /\n",rord);
line="END \n"; fprintf(scheme_file,"%s",line);
fclose(scheme_file);
}

```

3.2 input_genocop.dat file

In the sample file each line is defined by a coment. Note the line numbering is only for only added for convenience. Genetic operators (GOp) selection need some further explainations:

Each interger refers to the number of individuals involved in a GOp. The selection order is defined by the list of the GOp below. There are two important points to keep in mind:

- For **crossover GOp**, the number of individuals involved must be a multiple of 2 since 2 parents are needed to produce an offspring.
- The sum of the number of individuals involved must be stricly inferior to the half of the population size.

The selection order is:

1. **simple uniform mutation**
2. **boundary mutation**
3. **simple non-uniform mutation**
4. **whole arithmetical crossover**
5. **simple arithmetical crossover**
6. **whole non-uniform mutation**
7. **heuristic crossover:**

See Sec. 2.3 for a full definition of the GOp.

```

1   8      0      0      8 ! N_param      0      0      N_param
2
3   12.0    1      18.0      ! lower_bnd  param_#  upper_bnd
4   15000.0 2      35000.0     ! lower_bnd  param_#  upper_bnd
5   0.8     3      1.2      ! lower_bnd  param_#  upper_bnd
6   1.0     4      2.0      ! lower_bnd  param_#  upper_bnd
7   6.0     5      15.0     ! lower_bnd  param_#  upper_bnd
8   10000.0 6      20000.0     ! lower_bnd  param_#  upper_bnd
9   1.0     7      1.2      ! lower_bnd  param_#  upper_bnd
10  0.5    8      1.0      ! lower_bnd  param_#  upper_bnd
11
12  17      50    ! population size           generations number
13
14  0      0      1      2      2      1      2 ! genetic operators selection
15
16  0.10  ! selective pressure coeff , do not modify
17
18  0      ! minimisation=0 / maximisation=1
19

```

```

20   1      ! RESTART FLAG: multiple random=0, single random=1,
21           !                         single restart=2, multiple restart=3
22
23   6      ! parameter for non-uniform mutation, do not modify
24
25   10     ! parameter for simple cross over, do not modify
26
27   1      ! test case number, not used

```

3.3 input_eporck.dat file

The sample file comments describe the file structure. Note the line numbering is added only for convenience. This file is read via a parser. Any comment can be inserted. The numerical values are only read after recognition of the following key-words (all are finalised with :):

- Constraints-number: Constraints-indexes: Constraints-weights:
- OP-number:OP-weights:
- Target-values:

There is no input order since the file is rewinded for each keyword.

```

1  ****
2  *          INPUT DATA FOR EPORCK  V1.8      *
3  ****
4
5  -----
6  * Constraints section *
7  -----
8  Constraints-number:  4
9  Constraints-indexes: 1 2 6 7
10 Constraints-weights: 5.0 1.0 2.0 2.0
11 -----
12 * Operating Points section *
13 -----
14 OP-number: 3
15 OP-weights:      1.0 1.0 1.0
16
17 -----
18 * Reference values : 1 line per OP, 1 column per constraint *
19 -----
20 OP1:          S1          T2          Y_CO          Y_C02
21 Target-values: 5.877000e-01    1.779800e+03    3.073600e-05    7.718800e-02
22 OP2:
23 Target-values: 8.907000e-01    1.956100e+03    2.037000e-04    9.213200e-02
24 OP3:
25 Target-values: 1.172400e+00    2.113700e+03    9.499800e-04    1.058500e-01

```

This input file means that: The optimisation will use 4 constraints which are S_L (selector #1), T_{ad} (selector #2), the third species outlet mass fraction (selector #6) and the fourth species outlet mass fraction (selector #7). The α_ϕ weights coefficients (see Sec. 2.2) are respectively 5.0 1.0 2.0 and 2.0. Each scheme is evaluated with PREMIX on three operating points (OP). An OP is defined by the fresh mixture temperature and the mixture composition (defined in the equil.job file, see Sec. 3.5). The pressure is assumed to have the same value for all the OP. The α_k weights are all equal to 1.0. Finally the target values are tabulated.

3.4 restart.eporck file

This file is only needed when a single or multiple restart is specified in input_genocop.dat (see Sec. 3.2). For single restart, the information reduced to a unique line filled by the genes (floats separated by tabulation or space). For multiple restart, the files contains as much lines as a generation size. Each line defines an individual.

3.5 equil.job file

The user must only check the equil.job content. It must contain the same number of chained (CNTN) solutions (ie. same # of operating points) than declared in the input_eporck.dat file. At the initialisation, equil.job is read to generate the premix command file: premix.job. This, taking into account, pressure, inlet temperature, and mixture composition in order to setup temperature profiles, and chained premixed computation. Note that *mixture composition unit is mole fraction*.

equil.job:

```

HP
PRES 1.0
TEMP 300.0
REAC CH4    0.500
REAC O2     2.000
REAC N2     7.520
TEST 1800
CNTN
END
REAC CH4    0.800
REAC O2     2.000
REAC N2     7.520
CNTN
END
REAC CH4    1.100
REAC O2     2.000
REAC N2     7.520
END

```

3.6 Script shell files

The user must only check the Chemkin executables and databases paths.

equil.sh:

```

#!/bin/sh
# Path to Chemkin executables
CK=/local/chemkin36/bin
# Premix scheme name, DO NOT MODIFY
scheme=premix.scheme

```

```

# Swap chemout path to file or no output
#chemout=/dev/null
chemout=chem.out
chemlink=chem.asc
# Path to thermodynamic database file
chemdat=/local/chemkin36/data/therm.dat
# Equil command file name, DO NOT MODIFY
equilinput=equil.job
# Swap equilout path to file or no output
#equilout=/dev/null
equilout=equil.out
# Equil solution file name, DO NOT MODIFY
equilbin=equil.bin
# chemkin interpreter
$CK/chem -i $scheme -o $chemout -c $chemlink -d $chemdat
# Equil
$CK/equil -i $equilinput -o $equilout -b $equilbin -c $chemlink
exit

premix.sh:
#!/bin/sh
# Path to Chemkin executables
CK=/local/chemkin36/bin
# Premix scheme name, DO NOT MODIFY
scheme=promix.scheme
chemlink=chem.asc
# Swap tranout path to file or no output
#tranout=/dev/null
tranout=tran.out
# Path to transport database file
trandat=/local/chemkin36/data/tran.dat
tranlink=tran.asc
# premix command file, DO NOT MODIFY
job=promix.job
# Swap premixout path to file or no output
#premixout=/dev/null
premixout=promix.out
# Premix solution file, DO NOT MODIFY
savebin=save.bin
# tran
$CK/tran -o $tranout -d $trandat -c $chemlink -t $tranlink
# premix
$CK/promix -i $job -o $premixout -b $savebin -c $chemlink -t $tranlink
exit

```

4 Running EPORCK

4.1 Run check list

4.2 Getting started

The command line to launch EPORCK :

|prompt> **eporck_V1.6 input_genocop.dat genocop.out input_eporck.dat eporck.out**
Note that these file names can be user (re)defined.

The user can pause/continue or stop EPORCK before the normal program exit, using the folwing procedures:

- **PAUSE:** Simply create a file named **PAUSE_EPORCK** in the running directory. This easily done with the command: touch PAUSE_EPORCK

Note the program effectively pauses when the evaluation of the current generation is completed. So it may not be instantaneous. Check if eventually an old **GO_EPORCK** file is in the directory. If any, delete it before. If you do not delete it, the program pauses but restart within a short time.

- **CONTINUE:** To continue the run after a pause:

- Delete the **PAUSE_EPORCK** file.
- Create a file named **GO_EPORCK**. The program will restart within a minute.

- **STOP** Simply create a file named **STOP_EPORCK** in the running directory. This easily done with the command: touch STOP_EPORCK
This feature allows to stop properly EPORCK before the maximum number of generations, in order to complete all output files properly.

4.3 Output files

The **genocop.out** file (see Sec. 4.2) contains convergence information. Each time a new individual is the best, a line is appended. It contains generation number and cost function value.

```
Fri Jul 25 11:22:40 2003
Equalities :
Inequalities :
Domains :
  15.00 <= X1 <= 25.00
 15000.00 <= X2 <= 40000.00
  0.40 <= X3 <= 2.00
  0.40 <= X4 <= 2.00
  6.00 <= X5 <= 15.00
 5000.00 <= X6 <= 20000.00
  3.00 <= X7 <= 12.00
 5000.00 <= X8 <= 40000.00
  0.00 <= X9 <= 3.00
  0.00 <= X10 <= 3.00
Test case number : 1
Number of operators : 4 0 4 2 4 1 2
Number of generations : 100
Population size : 40
Parameter B : 6
Parameter Q : 0.100000
*****USING RESTART*****...
USING SINGLE POINT INITIAL POPULATION...
Generation#      Solution Value
Generation:      1          1.27352941
Generation:      2          0.35373634
Generation:      3          0.28377339
Generation:      8          0.28145617
Generation:      9          0.28115043
Generation:     10          0.25018328
|EPROCK> WARNING, PAUSE UNDER USER REQUEST...
|EPROCK> TO GO AHEAD, TYPE "touch GO_EPORCK" IN RUNNING DIRECTORY
|EPROCK> THINK TO REMOVE PAUSE_EPORCK BEFORE !
|EPROCK> WARNING, GO AHEAD UNDER USER REQUEST...
Generation:      21         0.18954715
Generation:      30         0.18917012
```

```

Generation:      31          0.16416746
Generation:      35          0.16402860
Generation:      39          0.14888197
Generation:      41          0.14788626
Generation:      42          0.14654692
Generation:      43          0.13493264
Generation:      49          0.12382746
Generation:      51          0.12382442
Generation:      57          0.12312602
Generation:      60          0.12272412
Generation:      67          0.12264616
Generation:      82          0.12264499
Generation:      83          0.12261784
Generation:      86          0.12261014
Generation:      88          0.12259977
Generation:      91          0.12259902
Generation:      94          0.12259635
Generation:      98          0.12259579
Generation:     100          0.12259553

```

Best solution was found at generation 100 (solution value = 0.12259553)

Best solution found:

```

X[ 1] :      18.39402390
X[ 2] :      28957.59765625
X[ 3] :      1.09906709
X[ 4] :      1.56450987
X[ 5] :      9.85662270
X[ 6] :      16572.35156250
X[ 7] :      7.22574282
X[ 8] :      32858.76171875
X[ 9] :      1.13119853
X[10] :      0.90438539

```

|EPORCK> Corresponding scheme file written (premix.scheme)

Total run time : 5185 seconds

The **eporck.out** file contains a run summary and information about each individual evaluation. Essentially used for debugging.

```

|EPORCK V1.8> premix.job file does not exists, generating premix.job...
*****
***** EPORCK SESSION *****
* Evolutionary Procedure for the *
* Optimisation of Chemical Kinetic schemes *
* VERSION V1.80 *
*****

```

|EPORCK V1.8> RUN SUMMARY:

|EPORCK V1.8> Number of constraints: 7

|EPORCK V1.8> Number of operating points (mixture composition): 4

|EPORCK V1.8> Index of premix output selected as target criteria:

- Criterion: 1 selected
- Criterion: 2 selected
- Criterion: 3 selected
- Criterion: 6 selected
- Criterion: 7 selected
- Criterion: 8 selected
- Criterion: 9 selected

|EPORCK V1.8> Response weights sum to 29.000000 , normalizing...

|EPORCK V1.8> Response's weight :

- Weight[1]: 0.413793
- Weight[1]: 0.413793
- Weight[2]: 0.137931
- Weight[3]: 0.034483
- Weight[4]: 0.034483
- Weight[5]: 0.034483
- Weight[6]: 0.310345

```

    - Weight[7]: 0.034483
|EPORCK V1.8> OP weights sum to 12.000000 , normalizing..
|EPORCK V1.8> OP's weights :
    - Weight[1]: 0.333333
    - Weight[2]: 0.166667
    - Weight[3]: 0.166667
    - Weight[4]: 0.333333
|EPORCK V1.8> Target responses (operating point = OP):
    - OP #1
        - Target response #1 = 0.587700
        - Target response #2 = 1779.800049
        - Target response #3 = 0.000580
        - Target response #4 = 0.000031
        - Target response #5 = 0.077188
        - Target response #6 = 0.063805
        - Target response #7 = 0.000004
    - OP #2
        - Target response #1 = 0.890700
        - Target response #2 = 1956.099976
        - Target response #3 = 0.000441
        - Target response #4 = 0.000204
        - Target response #5 = 0.092132
        - Target response #6 = 0.075733
        - Target response #7 = 0.000014
    - OP #3
        - Target response #1 = 1.172400
        - Target response #2 = 2113.699951
        - Target response #3 = 0.000373
        - Target response #4 = 0.000950
        - Target response #5 = 0.105850
        - Target response #6 = 0.087250
        - Target response #7 = 0.000049
    - OP #4
        - Target response #1 = 1.403600
        - Target response #2 = 2246.399902
        - Target response #3 = 0.000337
        - Target response #4 = 0.003192
        - Target response #5 = 0.117080
        - Target response #6 = 0.098111
        - Target response #7 = 0.000145
|EPORCK V1.8> Check PREMIX job file parameters:
|EPORCK V1.8>     - NPTS = 5
|EPORCK V1.8>     - XSTR = -1.50 cm
|EPORCK V1.8>     - XEND = 4.00 cm
|EPORCK V1.8>     - XCEN = 0.00 cm
|EPORCK V1.8>     - TFIX = 1073.0 K
|EPORCK V1.8>     - WMIX = 0.20 cm
|EPORCK V1.8>     END OF RUN SUMMARY
*****
|EPORCK V1.8> RUN HAS BEGUN AT : Fri Jul 25 11:22:40 2003
    Constraint # 0:
        OP#:0 R=0.0514974 Ropt=0.5877
        OP#:1 R=0.0667286 Ropt=0.8907
        OP#:2 R=0.0810181 Ropt=1.1724
        OP#:3 R=0.0939793 Ropt=1.4036
        Fitness[0]=2.59005
    Constraint # 1:

```

```

OP#:0 R=1769.43 Ropt=1779.8
OP#:1 R=1948.95 Ropt=1956.1
OP#:2 R=2113.67 Ropt=2113.7
OP#:3 R=2257.45 Ropt=2246.4
Fitness[1]=0.00419652
Constraint # 2:
OP#:0 R=0.00887623 Ropt=0.0005802
OP#:1 R=0.00730358 Ropt=0.00044103
OP#:2 R=0.00640637 Ropt=0.00037325
OP#:3 R=0.00585811 Ropt=0.00033685
Fitness[2]=2.80287
Constraint # 3:
OP#:0 R=2.28689e-05 Ropt=3.0736e-05
OP#:1 R=0.000171816 Ropt=0.0002037
OP#:2 R=0.000864963 Ropt=0.00094998
OP#:3 R=0.00313179 Ropt=0.0031922
Fitness[3]=0.148917
Constraint # 4:
OP#:0 R=0.0759388 Ropt=0.077188
OP#:1 R=0.0904783 Ropt=0.092132
OP#:2 R=0.104097 Ropt=0.10585
OP#:3 R=0.115009 Ropt=0.11708
Fitness[4]=0.0171888
Constraint # 5:
OP#:0 R=0.0645338 Ropt=0.063805
OP#:1 R=0.0769769 Ropt=0.075733
OP#:2 R=0.0891545 Ropt=0.08725
OP#:3 R=0.101003 Ropt=0.098111
Fitness[5]=0.0197823
Constraint # 6:
OP#:0 R=0.000306608 Ropt=3.8648e-06
OP#:1 R=0.000389734 Ropt=1.4029e-05
OP#:2 R=0.000416907 Ropt=4.8887e-05
OP#:3 R=0.000378165 Ropt=0.00014535
Generation 0 Fitness= 1.27353
etc ...

```

The **evaluation.log** file contains the evalutatation history. For each generation, an individual evaluation is summarised on a line by its:

- generation number
- the cost function evaluation
- genotype i.e the gene (parameter) sequence.

The **best.scheme** file is a CHEMKIN scheme format file updated at each improvement.

It describes the best scheme at the moment.

```

ELEMENTS
H O C N
END
SPECIES
CH4 O2 CO CO2 H2O NO N2
END
REACTIONS
CH4+1.502 => CO + 2H2O          2.477885E+18    0.0    29281.8
FORD / CH4 1.108466 /
FORD / O2 1.564027 /
CO + 0.502 = CO2                 7.491749E+09    0.0    14106.0
FORD / CO 1.000000 /
FORD / O2 0.500000 /

```

```

N2 + O2 => 2NO          1.047884E+09    0.88    60693.5
FORD / N2 1.803871 /
FORD / O2 0.620011 /
END

```

The **premix.scheme** file is a CHEMKIN scheme format file updated at the end of the run. It describes the best scheme of all.

4.4 Specific tools

These tools aim to facilitate the generation of constraints target values. The 1D laminar flames are computed with a complex chemical mechanism such as the GRI-Mech⁵ for methane, or natural gas. The user must be able to extract reference quantities such as: S_l , T_{out} , δ_L^0 , Y_k^{out} . Two essential tools help to extract these values from binary PREMIX V3.x solution file:

- **makeref_V1X:** This fortran tool reads the binary PREMIX file, and then edit an ASCII column file. It contains:
 - one line per operating point i.e for each equivalence ratio.
 - each line composed by one column per quantity. The quantities order is: ϕ , S_l , T_{out} , δ_L^0 and Y_k^{out} . For outlet species mass fractions, the order is the one declared in the premix scheme file.

There is a default input file called makeref.choices composed of three lines:

```

1   'save.bin' ! PREMIX binary solution file name
2   0.4        ! first equivalence ratio point
3   1.5        ! last equivalence ratio point
4   0.1        ! equivalence ratio step

```

- **extcol:** This script tool extracts selected columns of an ASCII file ignoring lines begining with the # character.
Example: The user wants to extract columns 1, 2, 17, and 5 from a file called "reference.dat" to a file called "myrun_ref.dat". He types in an UNIX shell window: **extcol reference.dat 1 2 17 5 > myrun_ref.dat**
- **pre2slt2:** A fortran tool that postprocess PREMIX solution and plot $S_l(\phi)$ and $T_{out}(\phi)$ in an xmgr window.
- **pre2yk2:** A fortran tool that postprocess PREMIX solution and plot Y_k^{out} in an xmgr window.

⁵http://www.me.berkeley.edu/gri_mech

5 Problem sample

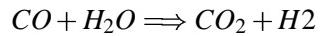
2-3 step-scheme ideas:

•

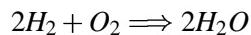
1.



2.



3.

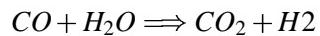


•

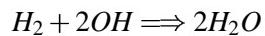
1.



2.



3.

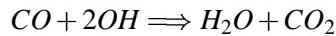


•

1.

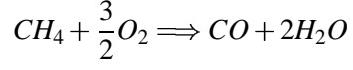


2.

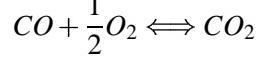


•

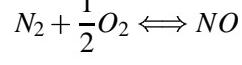
1.



2.



3.



• Jones & Lindstedt like:

1. $C_nH_{2n+2} + \frac{n}{2}O_2 \implies nCO + (n+1)H_2$
2. $H_2 + \frac{1}{2}O_2 \iff H_2O$
3. $CO + H_2O \iff CO_2 + H_2$

- 4-step with Zeldovich simplified mechanism #1:

1. $CH_4 + \frac{3}{2}O_2 \implies CO + 2H_2O$
2. $CO + O_2 \iff CO_2 + O$
3. $N_2 + O \iff NO + N$
4. $N + O_2 \iff NO + O$

- 4-step with Zeldovich simplified mechanism #2a:

1. $CH_4 + \frac{5}{2}O_2 \implies CO + 4OH$
2. $CO + 4OH \iff CO_2 + 2H_2O + O$
3. $N_2 + O \iff NO + N$
4. $N + O_2 \iff NO + O$

- 4-step with Zeldovich simplified mechanism #2b:

1. $CH_4 + 3O_2 \implies CO + 4OH + O$
2. $CO + 2OH \iff CO_2 + H_2O$
3. $N_2 + O \iff NO + N$
4. $N + O_2 \iff NO + O$